A New Approach to Optimal Cell Synthesis

Madsen, Jan

Published in:
Proceedings of the IEEE International Conference on Computer-Aided Design, ICCAD-89

Link to article, DOI:
10.1109/ICCAD.1989.76965

Publication date:
1989

Document Version
Publisher's PDF, also known as Version of record

A New Approach to Optimal Cell Synthesis

Jan Madsen

DesignCenter of Electronics Institute
Technical University of Denmark
DK2800 Lyngby, Denmark

Abstract

This paper presents a set of algorithms for optimal layout generation of CMOS complex gates. The algorithms are able to handle global physical constraints, such as pin placement, and to capture timing aspects. Results show that this new approach provides better solutions in area and speed compared to other methods. The algorithms have been implemented in a cell compiler (**CELLO**) working in an experimental silicon compiler environment.

1 Introduction

Cell compilers, which translate a behavioral cell description into mask layout, makes it possible to deal with much larger "cell libraries" than the usual standard cell approach, by replacing many primitive gates, such as NAND and NOR gates, with a single *complex gate* [8]. This method reduces layout area and improves performance.

Algorithms pioneered by Uehara and vanCleemput [8], who introduced the idea of *Euler paths* and *dual trails* to obtain a layout in the line-of-diffusion layout style, have concentrated on specifying a graph theoretical model of the problem and finding an optimal solution. [5] generalized the work of [8] and proposed an optimal method based upon the same cell model. However they did not address the problem of reordering transistors in the netlist, which may be a considerable limitation as the order influences area as well as electrical performance. [3] propose an algorithm, which does not restrict transistor pairs to be aligned. Though this might have some advantages when trying to chain different cells, the cell height will increase due to a more complex routing within the cell. [2] propose an algorithm based upon the *delayed binding* concept, which produces a minimum width layout, i.e. an optimal transistor ordering. But as will be shown in the next section, minimum width does not imply minimum area. Common to these algorithms is, that an optimal solution is the one giving the smallest possible cell layout area, i.e. a local optimum.

If, however, a global optimal solution is to be realized, global constraints have to be taken into account, e.g. the pin placement.

So in order to fully utilize the flexibility of the cell compiler concept, it should be possible to handle pin placement as a constraint.

This paper presents a set of algorithms able to handle a constrained pin placement for global optimization and still produce local optimal cell layout if possible.

2 General Design Aspects

Using the line-of-diffusion layout style [8], where physically adjacent transistors may share a common diffusion area, an optimal transistor ordering will maximize the number of shareable diffusion areas and minimize the number of diffusion gaps.

However, a given transistor circuit might have several optimal orderings, all giving a minimal-width layout, but resulting in different heights. Figure 1 shows the layout of three different orderings of the same transistor circuit. Comparing the layout areas, it is obvious that an optimal ordering does not necessary lead to an area-optimal layout. Further, the choice of ordering significantly influences the circuit performance (i.e. speed) as shown in Table 1, where the worst case propagation delays for the three circuits in Figure 1 are listed.

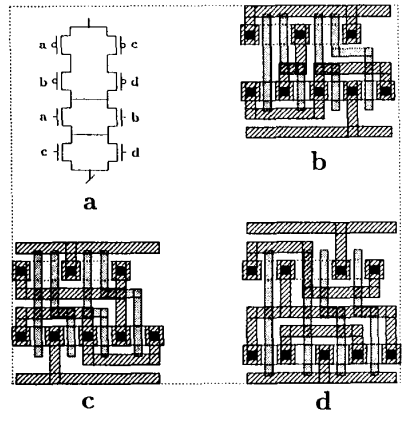

Figure 1: The effect of transistor ordering on layout area: a) transistor circuit; b) order: a b d c; c) order: d c a b; d) order: b d c a.

Delay	Circuit		
	b	c	d
$t_{pdL\rightarrow H}$	1.60	1.66	1.76
$t_{pdH\rightarrow L}$	0.68	0.73	0.75

Table 1: Worst case propagation delay (ns) for the circuits from Figure 1.

In the search for an optimal ordering, the *delayed binding* concept is often used, [2]. When using this concept, the transistor circuit netlist is regarded as changeable, as long as the function realized by the netlist is not changed. Though the concept of delayed binding may be efficient, one should be able to distinguish

CH2805-0/89/0000/0336$01.00 © 1989 IEEE

between fixed and changeable netlists, as the ordering also influences the circuit performance. Thus the netlist will be regarded as *fixed* if pre-timing optimization has been performed. Simulations [4] show that for a medium size complex gate (cf. Figure 3) a worst case delay gain of 22% may be obtained by choosing the right netlist configuration.

The mathematical model should be able to reflect these timing and area aspects, in order to produce a physically correct solution.

3 Model and Problem Formulation

A CMOS transistor circuit at the cell level is composed of a pull-up and a pull-down network. Each of these two networks can be represented by an undirected two-terminal multigraph $\mathcal{G}$, in which an edge (e) represents the drain/source connection/path of a transistor and a vertex (v) the net connecting several transistors. Each edge is indexed by the gate net (signal) of the corresponding transistor. The two terminals of the multigraph represents the power and output nets.

A transistor ordering, i.e. a path $\mathcal{P}$, which is an alternating sequence of vertices and edges, in which the edges are unique, found by the use of the eulerpath concept, corresponds to unfolding the pull-up and pull-down graphs. Consequently some of the vertices are split into one or more vertices, which has to be connected by internal routing. The way the graph is unfolded and the endpoints of the path are chosen, influences the final layout as shown in Figure 1.

In order to deal with this, all vertices are classified into one of four categories reflecting the characteristics of the layout. A vertex in a *two-terminal multigraph* is either a *terminal-vertex* (T) or an *internal-vertex* (I). If a terminal vertex is connected to the power supply, it is classified as a *power-terminal* (P), and if it is connected to the output, it is classified as an *output-terminal* (O). An internal vertex v_i is classified as either a *single-vertex* (S) or a *multiple-vertex* (M), determined by the valence $^\circ v_i$ of the vertex v_i being equal to or greater than 2.

4 Algorithms

To solve the problems described above, a set of algorithms working on the two-terminal multigraph $\mathcal{G}$ is formulated.

EulerPathExist

This algorithm examines eulerpath existency based on the theorem:

Theorem 1 *A graph $\mathcal{G}$ has an eulerpath iff it is connected and only two vertices (or no vertex) in $\mathcal{G}$ have odd valence. The path $\mathcal{P}$ $\{v_1, e_1, v_2, e_2, \ldots, v_n, e_n, v_{n+1}\}$ is called an open (closed) eulerpath in $\mathcal{G}$ if $v_1 \neq v_{n+1}$ ($v_1 = v_{n+1}$).*

FindSubGraphs

This algorithm finds the complete set, $\mathcal{S}_{\mathcal{G}_{sub}}$, of two-terminal serial/parallel subgraphs $\mathcal{G}_{sub}$ in $\mathcal{G}$. Subgraphs are used when reordering the edges in the graph to meet given constraints. The set of subgraphs is deduced by the use of an recursive graph reduction procedure, in which a number of edges are transformed into a single edge representing the subgraph. In each reduction step there are two types of two-terminal subgraphs, serial ($\mathcal{G}^s_{sub}$), and parallel ($\mathcal{G}^p_{sub}$), according to the connection of the edges.

A *first-order* subgraph is a subgraph not containing any other subgraph, i.e. only edges corresponding to transistors.

4.1 Constrainted Pin Placement

If the pin placement is given as a constraint, then depending on whether or not the netlist is fixed, two algorithms can be formulated, which traverse the graphs in order to match the pin placement $\mathcal{P}_{pin}$ to one of the possible eulerpaths.

FindSpecificEulerPath

This algorithm traverses a fixed graph following the path $\mathcal{P}_{pin}$ given by the pin placement. In this case a match of the path to one of the possible eulerpaths cannot be guaranteed. So if a match is not possible, one or more diffusion gaps must be accepted or the given pin placement must be ignored.

ChangeEulerPath

This algorithm performs eulerpath preserving changes on the graph in order to match a given path $\mathcal{P}_{pin}$. The algorithm traverses the graph as in **FindSpecificEulerPath**. When the next edge from the pin placement can not be reached directly, an edge interchange has to take place before the path trace can continue. There are two possible interchanges, which do not change the functionality: Edge *interchange* within a serial two-terminal subgraph $\mathcal{G}^s_{sub}$ and a two-terminal subgraph *swap* defined as:

$$\mathcal{G}_{sub}(v_i, v_j) \Rightarrow \mathcal{G}_{sub}(v_j, v_i) \tag{1}$$

In the first case, let $\mathcal{S}_{temp}$ be the set of edges connected to the previous edge e_i. If the next edge e_{i+1} belongs to a serial two-terminal subgraph of $\mathcal{G}$, which is connected to e_i, that is

$$\exists e_m \in \mathcal{S}_{temp}, \exists \mathcal{G}^s_{sub} \in \mathcal{S}_{\mathcal{G}_{sub}} : \{e_m, e_{i+1}\} \subseteq \mathcal{G}^s_{sub} \tag{2}$$

then the edge e_{i+1} and e_m are interchanged such that e_{i+1} will follow the previous edge e_i as required by the pin placement.

In the second case where the next edge does not belong to a serial-subgraph, the problem is far more complicated. First a possible candidate for a subgraph swap has to be found. Depending on whether the path traced so far $\mathcal{P}_{path}$ equals one of the subgraphs or not, there exists two possible ways of choosing a candidate. Let $\mathcal{P}_{path} = \{\ldots, e_i\}$ and $\mathcal{P}_{pin} = \{\ldots, e_i, e_{i+1}, \ldots\}$, where the edges are defined as $e_i(v_n, v_m)$ and $e_{i+1}(v_k, v_l)$, and let $\mathcal{S}_{\mathcal{G}_{sub}}$ be the total set of subgraphs in $\mathcal{G}$, then if

$$\begin{aligned} &\exists \mathcal{G}_{sub}(v_i, v_j) \in \mathcal{S}_{\mathcal{G}_{sub}} : \\ &\quad (\mathcal{G}_{sub} \cap \mathcal{P}_{path} = \mathcal{P}_{path} \vee \mathcal{G}_{sub} \cap \mathcal{P}_{path} = \emptyset) \wedge \\ &\quad \{v_i, v_j\} \cap \{v_k, v_l\} \neq \emptyset \wedge \{v_i, v_j\} \cap \{v_n, v_m\} \neq \emptyset \end{aligned} \tag{3}$$

$\mathcal{G}_{sub}$ is a possible candidate for a subgraph swap. The first term selects the subgraph and ensures that the part of the graph which already has been traced will not be changed by the swap, while the second term ensures that e_i and e_{i+1} becomes neighbouring edges after the swap. Since the swap should preserve the existence of an eulerpath, it will only be accepted if

$$\begin{aligned} &f(^\circ_{\mathcal{G}_{sub}} v_i) = f(^\circ_{\mathcal{G}_{sub}} v_j) \vee \\ &f(^\circ_{\mathcal{G}_{sub}} v_i) \neq f(^\circ_{\mathcal{G}_{sub}} v_j) \wedge f(^\circ_{\mathcal{G} \setminus \mathcal{G}_{sub}} v_i) = f(^\circ_{\mathcal{G} \setminus \mathcal{G}_{sub}} v_j) \end{aligned} \tag{4}$$

where $^\circ_{\mathcal{G}_{sub}} v_j$ denotes the number of edges connected to vertex v_j, which belongs to the subgraph $\mathcal{G}_{sub}$, and $f(x)$ is 1 if x is odd and 0 if x is even. Figure 2 illustrates the two possible situations which

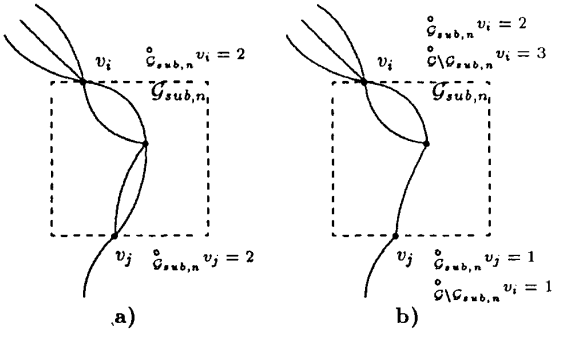

Figure 2: Subgraph swap acceptance; a) $f(^{\circ}_{\mathcal{G}_{sub}}v_i) = f(^{\circ}_{\mathcal{G}_{sub}}v_j)$; b) $f(^{\circ}_{\mathcal{G}_{sub}}v_i) \neq f(^{\circ}_{\mathcal{G}_{sub}}v_j) \wedge f(^{\circ}_{\mathcal{G}\setminus\mathcal{G}_{sub}}v_i) = f(^{\circ}_{\mathcal{G}\setminus\mathcal{G}_{sub}}v_j)$.

lead to swap acceptance. If no swap is accepted, then either the pin placement has to be ignored (terminating the algorithm) or a diffusion gap has to be inserted. If no candidate is found, a series of swaps denoted as a *multi-swap* is performed. A multi-swap is aimed at bringing e_i and e_{i+1} together. While the swap acceptance is identical with a normal swap, only subgraphs where $\mathcal{G}_{sub} \cap \mathcal{P}_{path} = \emptyset$ are choosen as candidates, as the traced path would otherwise be changed during the swaps.

Since it is possible to control the properties of the vertices while rearranging the graph, the algorithm can be used to change the endpoint vertices (i.e. the second type of acceptance) in order to either obtain dual eulerpaths if one does not exist, or to influence timing performance. Further the algorithm can be used to redistribute parasitic capacitances in order to inprove timing performance, i.e. moving high valenced vertices further away from the output-terminal.

4.2 No Constrainted Pin Placement

When no pin placement is given the objective is to obtain an area optimal layout, taking into account timing aspects whenever possible.

MakeEulerPathExist

This algorithm is a special case of **ChangeEulerPath**, which performs function preserving changes on the graph in order to achieve eulerpath existency. All subgraphs $\mathcal{G}_{sub}(v_i, v_j)$ are possible candidates for a swap. The swap is only performed if the total number of odd valenced vertices are reduced, i.e. if

$$\exists \mathcal{G}_{sub}(v_i, v_j) \in \mathcal{S}_{\mathcal{G}_{sub}} : \quad (5)$$
$$f(^{\circ}v_i) = 1 \wedge f(^{\circ}v_j) = 1 \wedge f(^{\circ}_{\mathcal{G}_{sub}}v_i) \neq f(^{\circ}_{\mathcal{G}_{sub}}v_j)$$

When all subgraphs have been examined, then if at most two vertices have odd valence, the graph has been transformed into having an eulerpath.

FindEulerPath

This algorithm finds a dual eulerpath in the two dual graphs $\mathcal{G}^p$ and $\mathcal{G}^s$ (index p for parallel and s for serial) if such exist. Depending on whether or not the netlist is fixed, the algorithm is allowed to rearrange transistors in the two graphs in order to obtain an optimal solution.

First consider the case of a changeable netlist. In a parallel graph the branch to traverse can be selected arbitrarily, the selection corresponds to choose a certain order of elements in the corresponding serial graph. In the case of a changeable netlist the problem can be formulated as to select an order in every serial subgraph which leads to the existency of a dual eulerpath ([6]). Consequently the algorithm starts in the parallel graph $\mathcal{G}^p$, in order to delay the binding of the order of the serial graph.

To reduce the problem size all *first-order* subgraphs are reduced prior to the path tracing procedure. In $\mathcal{G}^p$ *odd* parallel and all serial subgraphs are substituted with a single edge, while *even* parallel subgraphs are removed. In $\mathcal{G}^s$ all first order subgraphs are substituted with single edges. *Even* serial subgraphs corresponding to even parallel subgraphs in $\mathcal{G}^p$ are marked. The reduced graphs of $\mathcal{G}^p$ and $\mathcal{G}^s$ are denoted $\mathcal{R}^p$ and $\mathcal{R}^s$ respectively.

Two essential properties influence the solution; the choice of endpoints in a closed eulerpath, and at which of the two possible vertices in $\mathcal{R}^p$ to resubstitute a *even* parallel subgraph. Other algorithms [2] seems to make an arbitrary choice, but as shown in Figure 1, the choice may influence both area and electrical performance. Therefore a set of rules is proposed for an optimal choice of the endpoint in a closed path. In order of preference:

Rule 1: Terminal-vertex. Prefer power-terminal to output-terminal. A terminal vertex (especially the power-terminal) will very likely require one routing track. Consequently the splitting of such a vertex will not influence the layout area. Further, the extra amount of diffusion area from the vertex splitting will have the least influence on the electrical performance when added to a power-terminal, while having significant influence when added to a output-terminal (the most sensitive vertex).

Rule 2: Internal-vertex. Always choose a multiple-vertex. If a single-vertex is chosen, cell height is increased since the splitting of a single-vertex causes two tracks to be used because of power and output obstructions (cf. Figure 1). Also, the electrical performance is influenced as the parasitic capacitance is significantly increased.

Based upon the theorem:

Theorem 2 *Having two dual graphs $\mathcal{G}^p$ and $\mathcal{G}^s$, a necessary condition for the existence of a dual eulerpath is that the same edges can be reached from endpoints in both $\mathcal{G}^p$ and $\mathcal{G}^s$.*

it is possible to decide whether or not there is a chance of finding a dual eulerpath. If not, and the netlist is changeable, endpoints may be changed to meet this condition as described in **ChangeEulerPath**.

The algorithm first finds all possible paths between the two endpoints in $\mathcal{R}^p$. This number is usually very limited and heuristics are used to remove unfeasible paths. In order to decide at which of the possible vertices to resubstitute *even* parallel subgraphs, the dual graph ($\mathcal{R}^s$) is traversed for each of the feasible paths ($\mathcal{P}^p$) from $\mathcal{R}^p$. When the next edge from $\mathcal{P}^p$ cannot be reached directly, either a subgraph swap as described in **ChangeEulerPath** has to take place or one of the special marked edges has to be used. Using one of the special marked edges corresponds to choose the vertex at which to resubstitute a *even* parallel subgraph in $\mathcal{G}^p$. If neither of the two situations leads to a solution, the next path is tried.

In the case of a fixed netlist swaps are not allowed. Each path is completly traced and whenever the next edge cannot be

reached directly or by a special marked edge, a gap is inserted. Finally the best solution is chosen.

Once a solution has been found all *first-order* subgraphs can be resubstituted. If possible, high valenced vertices are placed away from the output vertex in order to improve performance.

5 Results

The algorithms described in this paper have been used to implement a cell compiler (**CELLO**) working in the open experimental silicon compiler environment **CATOE**[1] [1]. **CELLO** produces a symbolic layout, which is then transformed into mask layout by a compactor. Figure 3 shows **CELLO** generated layout after compaction, together with the same circuit realized by two other methods [8] and [2]. In order to make this comparison, the same physical interpretation of the paths was used to generate the layouts for the different algorithms. Table 2 lists the area parameters for different layouts, which shows that the new approach presented in this paper performs better than [8] and [2]. Circuit C3 corresponds to the layouts of Figure 3.

The first test chip generated by means of the **CATOE**-system has been submitted for fabrication in a 2μ n-well CMOS process[2]. Cell synthesis was performed using **CELLO**. Figure 4 shows a section of the chip core.

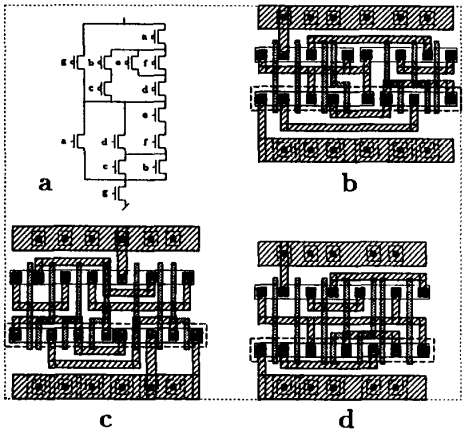

Figure 3: Layout results using different algorithms; a) transistor circuit; b) [8]; c) [2]; d) **CELLO**.

Circuit	#trans.		Algorithm		
			[8]	[2]	CELLO
C1	16	W	77	67	63
		H	71	83	71
C2	12	W	54	63	54
		H	59	59	59
C3	14	W	67	67	58
		H	71	77	71
C4	22	W	95	97	87
		H	71	95	77

Table 2: Width and height of layouts generated by 3 different algorithms.

[1]System developed by the **CATOE**-group (Computer Aided TOol Engineering) at Electronics Institute.

[2]The chip is produced under the Nordic brokerage service NORCHIP

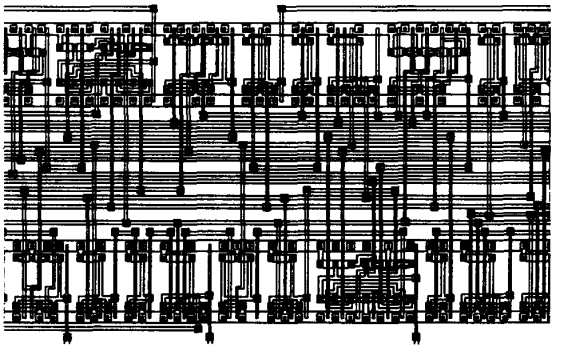

Figure 4: Section of the first chip generated by the **CATOE**-system. The individual cells are generated from netlist descriptions by **CELLO**.

6 Conclusion

A new approach for optimal layout generation of CMOS complex gates has been presented. This approach handles global constraints such as pin placement, given by a pre-place and route step, and timing considerations, by distinguishing between fixed and changeable netlists. Further, the approach has been used to implement a cell compiler **CELLO**, which provides better solutions than previous algorithms.

References

[1] A.C. Andersen, J. Madsen, J.R. Madsen and H. Pallisgaard, *"A Dynamic Environment for VLSI Design Tools"*, NORSILC/NORCHIP seminar '88, Copenhagen 26-27 oclober, 1988.

[2] C.Y. Roger Chen and Cliff Yungchin Hou, *"A New Layout Optimization Methodology for CMOS Complex Gates," IEEE ICCAD-88.*, pp.368-371, 1988.

[3] M. Lefebvre and C. Chan, *"Optimal Ordering of Gate Signals in CMOS Complex Gates: A Heuristic Approach,"* internal paper, Dept. of Electronics, Ottawa, Canada, 1988.

[4] J. Madsen, *"The Impact of Gate Ordering on Circuit Delay,"* internal paper, Designcenter of Electronic Institute, Technical University of Denmark, EI-LHT148, 1988.

[5] R.L. Maziasz and J.P. Hayes, *"Layout Optimization of CMOS Functional Cells," 24th ACM/IEEE Design Automation Conference.*, pp.544-551, 1987.

[6] R. Nair, A. Bruss and J. Reif, *"Linear Time Algorithms For Optimal CMOS Layout," VLSI: Algorithms and Architectures,.*, Bertollazi and Loccio Eds, North-Holland Pub., pp.327-338, 1985.

[7] Y. Shiraishi, J. Sakemi, M. Kutsuwadw, A. Tsukizoe and T. Satoh, *"A High Packing Density Module Generator for CMOS Logic Cells," 25th ACM/IEEE Design Automation Conference.*, pp.439-444, 1988.

[8] T. Uehara and W.M. vanCleemput, *"Optimal Layout of CMOS Functional Arrays," IEEE Trans. Computer* Vol.c-30, pp.305-312, May 1981.

[9] S. Wimer, R.Y. Pinter and J.A. Feldman, *"Optimal Chaining of CMOS Transistors in a Functional Cell," IEEE Trans. on CAD,* Vol.CAD-6, no.5, pp.795-801, 1987.